Fall 12-6-2013

CMOS Smart Camera with Focal Plane Neighborhood-Parallel Image Processing

Joseph A. Schmitz
University of Nebraska – Lincoln, josephschmitz@unl.edu

CMOS SMART CAMERA WITH FOCAL PLANE NEIGHBORHOOD-PARALLEL IMAGE PROCESSING

by

Joseph Schmitz

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of Professors Sina Balkir and Michael Hoffman

Lincoln, Nebraska

December, 2013

CMOS SMART CAMERA WITH FOCAL PLANE NEIGHBORHOOD-PARALLEL IMAGE PROCESSING

Joseph Schmitz, M.S.

University of Nebraska, 2013

Advisors: Sina Balkir and Michael Hoffman

A neighbhorhood-based smart camera architecture is designed and implemented in a 0.13 µm CMOS technology. Each 8×8 region of pixels contains a small processor with local memory, which are tiled to form a full-resolution camera. Each processor operates in parallel, enabling high-speed image processing suitable for tracking and recognition tasks. The architecture features the programming flexibility of designs using chip-level and row-level processors while preserving the scalability of pixel-parallel processing elements. The neighborhood processors are implemented physically between the pixel photodiodes, creating multiple design challenges that are discussed in detail.

ACKNOWLEDGMENTS

I thank Professor Sina Balkir and Professor Michael Hoffman for mentoring me during my time at the University. Their teaching, flexibility and enthusiasm made my graduate experience both enjoyable and fruitful. I also learned a great deal from the other graduate students in the department, specifically Mahir Gharzai and Dan White, who not only assisted me with this research, but were also delightful company in the lab. Nathan Schemm also provided invaluable help reviewing the chip layout and sharing his design insights. Finally, I am grateful to my family who loves and supports me unconditionally.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of CMOS Imaging

Standard digital camera systems utilize a CCD or CMOS image sensor chip coupled to a processor to capture images. CMOS sensors originally had a relatively low SNR compared to CCD technologies, but this was solved by adding individual transistor amplifiers to each pixel called active pixel sensors described in [1]. Addition of a small memory and ADC at each pixel enabled pixel-parallel image acquisition in the 10,000 FPS regime [2]. The versatility of CMOS technology makes this possible since it allows for the construction of both analog and digital circuitry on the same chip.

The practice of inserting additional circuitry on CMOS imagers expanded to include more complex analog and digital circuits that perform image processing tasks without the intervention of an external processor. Early work on these vision chips focused on pixel-level processing that performed basic early vision tasks such as edge detection, noise reduction and other convolution-based algorithms. These were implemented using analog circuits, which were much smaller than the digital equivalents at the time. This maximized the ratio of the photosensor to the pixel area, called the

fill factor, which is associated with improved imaging performance. Since the analog circuits had limited reconfigurability, each vision chip was designed to perform a specific processing task that was fixed at the time of manufacture.

Over time, advances in CMOS technology reduced transistor dimensions, making it practical to use digital circuitry for vision chips. Pixel-parallel digital logic provided similar functionality to their analog counterparts, but offered an increased degree of configurability and programmability [3, 4]. This also allowed the implementation of higher-complexity algorithms such as compression, tracking and object recognition [4–7]. Many of these systems are partially or fully programmable, which allows the same design to perform multiple image processing tasks as required by the application [3, 4].

Smart cameras are a subcategory of vision chips which are designed to output extracted information instead of raw pixel data, such as the trajectory of a projectile or the classification of a recognized object in a scene. The high level of integration of smart cameras has performance, power and price advantages compared to standard cameras.

1.2 Parallel Processing in Smart Cameras

Low-level processing such as edge detection may be performed in a pixel-parallel manner by implementing a processing element (PE) in each pixel. This method of computation requires constant time regardless of the resolution. Area constraints limit the types of processing these PEs may perform to low-level operations with adjacent pixel data. However, this topology excels at early vision applications such as edge detection and morphological operations [4]. The isolated nature of the PEs makes it less ideal for algorithms that require access to distant pixel data.

Row-level and column-level processing is used to address this issue and are used to perform mid-level algorithms. Unlike pixel-level PEs that only have access to nearby pixel data, these PEs may quickly access data at any location in their row or column, which spans across the entire imager. This is ideal for computing image statistics such as histograms and centroids. It also is fast at performing image transformations including mirroring and rotations [3]. Since one PE is required per row, it slows down with increasing resolution unlike pixel-parallel PEs. They also consume significant area that would otherwise be used to increase the camera resolution.

Chip-level processing handles complex global processing better than the lower-level methods since it has direct access to all pixel data in the imager, such as in [5]. However, it doesn't scale with resolution and becomes a processing bottleneck for high resolutions or framerates. These processors may be combined with the other levels of parallelization to mitigate these disadvantages [4, 6]. Figure 1.1 shows an example of a smart camera with pixel-level, row-level and chip-level processing.

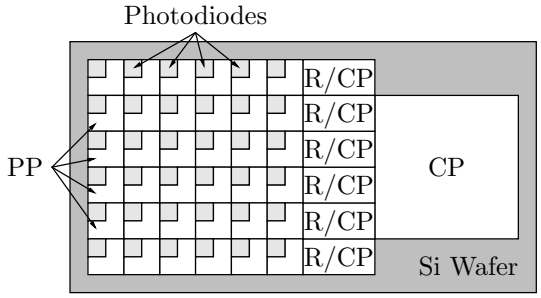


Figure 1.1: A hybrid smart camera architecture using three processing level

1.3 Motivation

In this work, a new category of neighborhood-parallel computation for smart cameras is introduced, designed and implemented for CMOS. Existing smart camera architectures are limited by a combination of programmability, scalability and power

consumption. To address these issues, a new architecture was developed that utilizes neighborhood-based processing. A neighborhood processor (NP) contains pixel ADCs, dedicated memory and an arithmetic logic unit (ALU) physically implemented within each 8×8 region of pixels across the entire camera. These NPs are tiled in a two-dimensional grid to form the desired resolution.

Image data may be processed locally as well as transmitted between adjacent NPs. A standard 8-bit instruction set allows implementation of generic algorithms that are executed in parallel across the camera. Once processing is complete, the resulting outputs are transmitted outside of the NP grid and off-chip. This architecture addresses several of the issues found in existing smart cameras.

1.3.1 Scalability

Similar to pixel-parallel architectures, each NP is responsible for a fixed number of pixels. For parallel algorithms, this enables constant processing time regardless of the camera resolution, shown in Figure 1.2. However, since they contain many pixels and cover a larger area, NPs may be more complex than standard pixel-level PEs. This allows for a level of computational power similar to row-level PEs without the scalability concerns. Finally, since NPs are implemented between the photodiodes, nearly all available chip area may be used to increase the camera resolution, unlike row and chip-level processors which may consume more than half the die [6–8].

1.3.2 Programmability

Many vision chips use specialized digital circuits to perform application-specific tasks such as convolutions [8, 9]. This architecture has minimal specialized hardware to allow for increased algorithm flexibility. It implements a generic 8-bit processor with

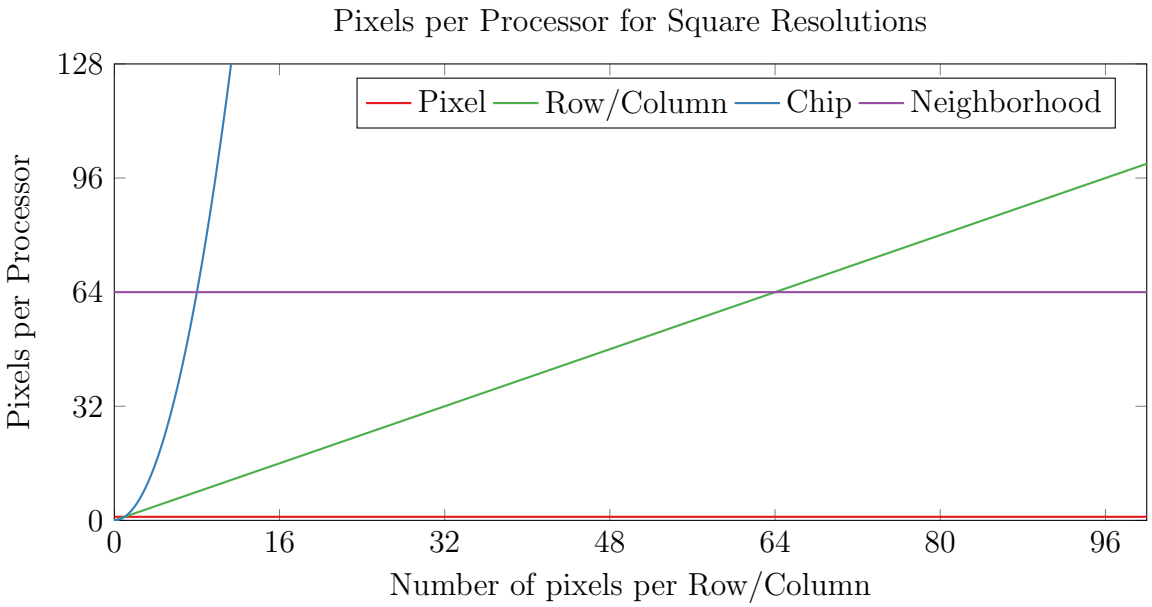


Figure 1.2: Scalability of neighborhood-level processing compared to other techniques

a simple assembly language and instruction set. A wide range of algorithms may be implemented for the design, including compression, tracking and recognition tasks similar to those found in [10–12].

1.3.3 Power Consumption

The architecture takes advantage of the spatial correlation of captured images, allowing for efficient computation of block-level filters and transforms often used in image processing applications [13, Chapter 15]. In typical scenes, objects of interest tend to be localized in space, so most of the pixels are not utilized until the object moves within the scene. Since NPs are inherently regional, only NPs corresponding to the object's current location need to kept active, and the rest may be turned off for power savings. This removes the need for specialized block-readout techniques like those found in [14].

1.3.4 Design Trade-offs

NPs strike a balance between computation speed and programmability. True application-specific pixel-parallel cameras are much faster than NPs since each pixel has dedicated hardware unlike the NP which shares memory and processing between 64 pixels. However, these architectures have fixed or restricted functionality that prevents them from implementing generic algorithms in software, unlike NPs. Cameras with chip-level processors offer the highest-degree of algorithm flexibility with feature-rich instruction sets and access to the pixel data for the entire imager. NPs maintain a high-degree of programmability while sacrificing access to distant pixel data, which allows for significantly better scalability than these chip-level processors. For many imaging algorithms, full-image availability is generally not necessary, and the NP's block-level approach is effective.

Chapter 2

Smart Camera Architecture

2.1 Comparison to Original Architecture

The NP architecture in this work is a based on a previous design from [15] with several revisions. It features a similar structure with a single global control unit (GCU) that decodes program instructions and controls a grid of NPs. Figure 2.1 shows a simplified block diagram of the architecture.

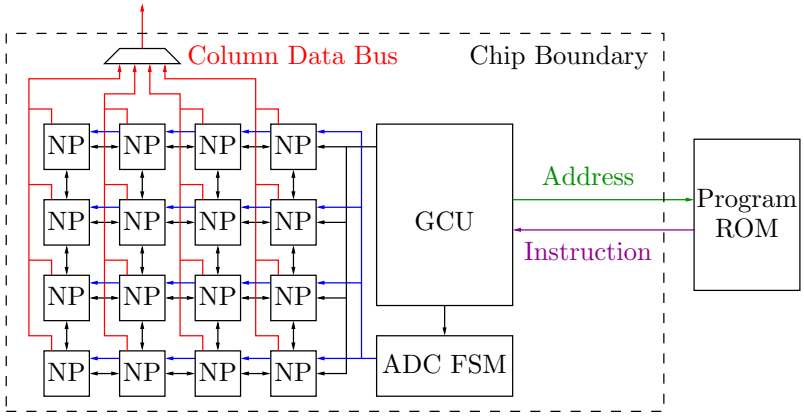


Figure 2.1: NP architecture block diagram

The original design used scripts to generate flat netlists for different NP grid sizes, but the new design is fully hierarchical and parameterized, so the resolution

may be easily altered. A rewrite of the HDL using generic RTL constructs gives the synthesis tools maximum flexibility when optimizing the design for speed and area. Specifically there are fewer modules in the new design since the synthesis tools can perform only limited optimization across module boundaries. The multiplexers used to access memory in the NP are now implemented with logic instead of tristate buffers in order to improve speed and area. For large busses, each tristate must drive a significant capacitive load since they cannot be buffered, requiring large, high-drive-strength tristates to be used. Logic-based multiplexers do not suffer from this problem since the capacitive load is distributed among the individual gates and digital buffers may be used to further improve timing. The overall block diagram of the revised NP is shown in Figure 2.3.

2.1.1 Added Features

In addition to changes in the RTL coding style and synthesis of the NP, it now has several new features to support more complicated algorithms. These features improve available memory, code density and speed.

In each NP, there are now 4 registers instead of 2 and any one of them may be used for indirect memory addressing. The memory size has increased from 192 bytes (3 per pixel) to 200 bytes. These extra bytes are used for storing temporary variables for algorithms that require the rest of memory to store intermediate pixel data.

The assembly language is rewritten to expand the number program flow instructions, which leads to smaller code size and improved readability without a significant increase in implementation area. The number of conditions has doubled from 4 to 8, which improves code density. Before it was necessary to read the status register flags and mask them to perform conditional operations, but this is now done in a sin-

gle instruction. This is especially important for creating high-efficiency loops which execute conditional branch instructions repeatedly.

Each column data bus is now implemented using digital muxes instead of tristate muxes. This removes the risk that malformed code would cause multiple NPs to drive the bus simultaneously, drawing large currents and possibly damaging the chip. Dedicated row output enable signals in the new design greatly increase the readout speed. The original architecture required a long series of instructions to allow a row of NPs to output to the bus, which is now accomplished with a single instruction. For example, Listing 2.1 shows code that outputs image data from a single row of NPs using the original architecture. Listing 2.2 performs the same function, but on the new architecture. Furthermore, in the manufactured imager, the entire code block in Listing 2.1 would be replicated seven more times, whereas the new architecture would only require seven more `OUT` instructions to be added to the code in Listing 2.2.

Listing 2.1: Code for the original NP in [15] that outputs a captured image

```
U LOADA A S 00
U ORA I 20
U MOVA A S 00
U LOADA A S 03
U ANDA I F0
U LOADA A S 00
U XORA I 28
U MOVA A S 00
Z LOADA A S 00
Z XORA I 40
Z MOVA A S 00
U LOADA A S 00
U ORA I 20
U MOVA A S 00
U LOADA A A 00
U SPL OUTA
U LOADB I 3F
U IRAM LOAD 0R
U SPL OUTA
U SUBB I 01
U LOADA A S 02
Z JUMP 1 12
```

Listing 2.2: Code for the revised NP identical in function to Listing 2.1

```
#DEFINE NUM_PIXELS 64
#DEFINE i          R0

START:
    LDR     i, 0
OUTPUT_LOOP:
    OUT     [i], 0
    ADD     i, 1
    CMP     i, NUM_PIXELS
    BNE     OUTPUT_LOOP
```

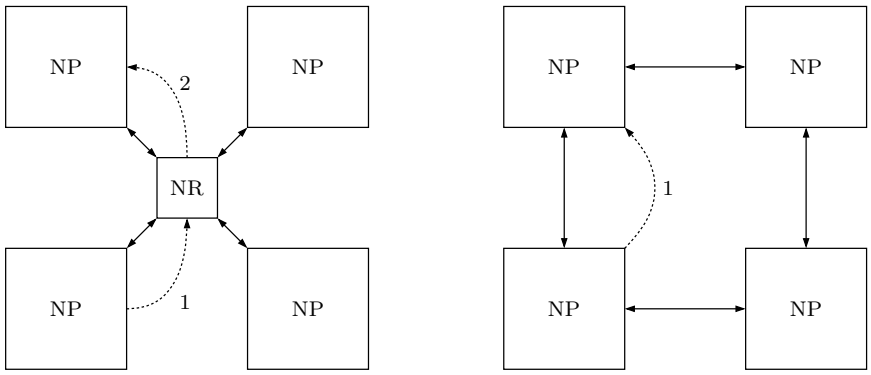
It is now possible to perform function calls, enabling the use of recursive algorithms and improving code reuse. To support this, the GCU contains a new 8 instruction program stack used to store return addresses. In this version, the maximum program size is 512 instructions, double the original. With minimal modifications and area penalty, this can be expanded using relative addressing to almost any required size. Sleep states are also improved in the new architecture. Similar to the original version, NPs may be conditionally turned on and off to save power. In addition to this, NPs now have multiple sleep depths that allow for nested conditional loops and other important programming structures.

In preparation for physical implementation on CMOS, the architecture now has a pixel ADC interface. A new finite state machine (FSM) in the GCU controls the timing for the pixel ADCs during image acquisition. It features programmable delays to accommodate different lighting environments. The ADCs use gray coding to improve performance, so the NP ALU now has a new instruction that converts gray code to its binary equivalent.

2.1.2 Removed Features

In order to save area, some features were removed from the architecture. Originally, NPs exchanged data through a neighborhood register (NR), which acted as a tem-

porary storage location accessible to adjacent NPs. In order to move data from one NP to another, two clock cycles were required. First data would be copied to the NR by the source NP. Second the destination NP would read the data from the NR, completing the transfer. In the new architecture, there are no NRs: NPs exchange data directly between one another's registers, which requires only one clock cycle to complete. Figure 2.2 shows this change.



(a) NP in [15] (b) NP in this work

Figure 2.2: Required instructions for data transfer in both NP architectures

The original design was able to perform 4-bit ALU and memory operations. This allowed three 8-bit registers to be used as two 12-bit registers for higher-precision computations. Since the implementation in this work uses 8-bit ADCs, this feature was not required and was removed to improve design area and routability.

There are no longer conditional ALU instructions. These are replaced by conditional sleep instructions, which perform the same function by preventing an NP from executing instructions while it is asleep. While sometimes slower, using sleep instructions requires less flag manipulation compared to the original architecture, and they are faster and easier to use in practice.

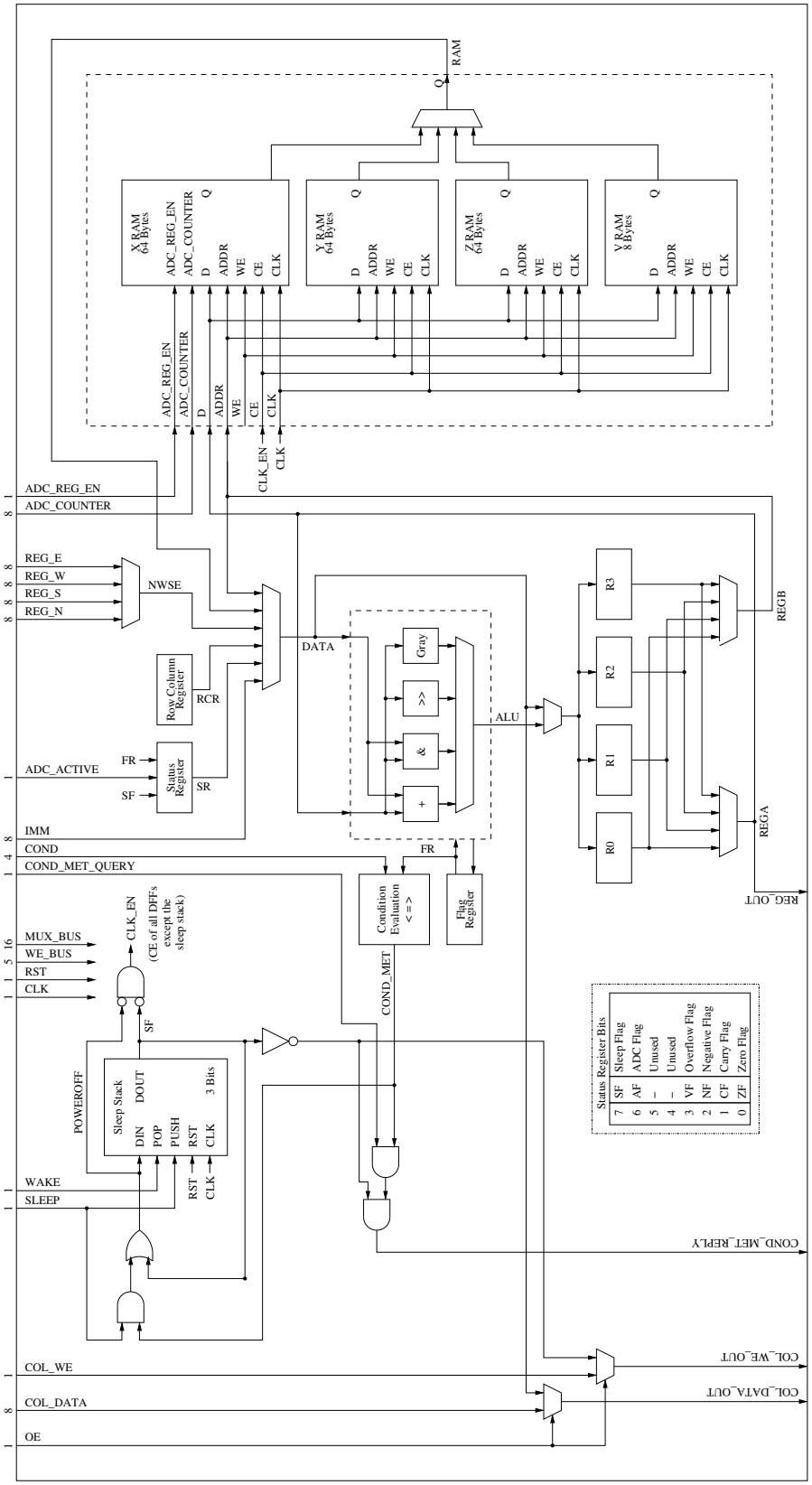


Figure 2.3: Block diagram of a single NP

Chapter 3

RTL Design

3.1 Overview

Chip design software from Cadence provides the necessary tools to implement a test chip for the NP architecture described in Chapter 2. As inputs, it requires a netlist that contains RTL descriptions of all digital logic along with abstractions that represent analog blocks, such as photodiodes. A commercial 0.13 µm, 8 metal design kit provides digital standard cells and IO pads for the chip. Hierarchical VHDL describes the logic and connectivity at each level of the design.

3.2 NP

The NP implementation closely follows the block diagram shown in Figure 2.3. One exception is that the input signals from the GCU to the NP are buffered and propagated through each NP. Also, the "condition met reply" signal from the NP back to the GCU is logically ORed with the reply from the previous NP and then buffered along to the next NP in the column. With these modifications, all local and global sig-

nals propagate through tiled NPs automatically, except for the clock, which requires special consideration for acceptable performance.

The VHDL implementation is largely technology-independent to aid in testing on FPGA before moving to an ASIC. However, several synthesis directives are added to force specific RTL constructs to be implemented with specialized standard cells instead of generic logic. This technique is used with multiplexers to reduce routing congestion in the NP at the cost of increased area, as shown in Figure 3.1. Even with these directives, the VHDL description is still directly portable to FPGA structures as seen in Chapter 6.

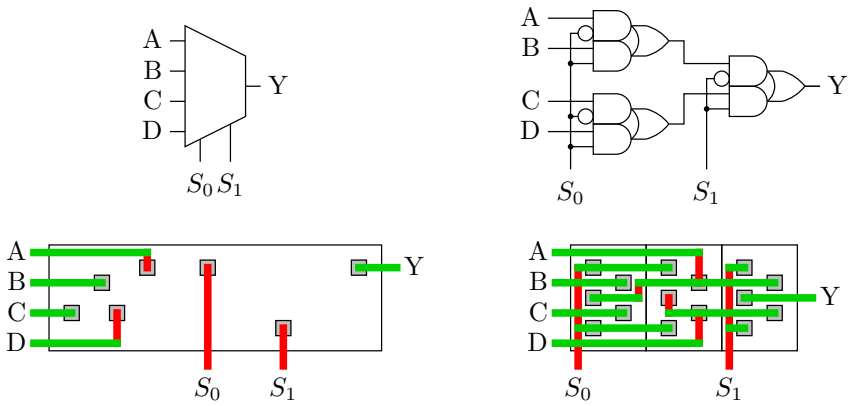


Figure 3.1: Multiplexer area and routing requirements

3.3 GCU

The GCU is modified to incorporate the FSM for the pixel ADCs. It contains two 16-bit registers that store the photodiode reset and integration times. These are output to the ADC FSM module where they are compared against internal timer counters to generate the output control signals required by the ADCs.

3.4 Test Structures

The RTL netlist includes additional structures that are not required for normal operation, but simplify testing and debugging procedures.

A MSP430-compatible microcontroller, called an NS430, is included on-chip and is inserted into the netlist to connect to IO pads and other tests structures. This microcontroller is implemented separately from the rest of the chip and then inserted into the layout at the end of the design flow. NPs are designed to operate independently of the NS430 and no algorithms are performed with it, as this would limit scalability of the architecture. However, it may be configured to interface with the NPs, providing instructions to the GCU and reading the outputs from the column data buses. This creates a single-chip solution that requires no external parallel program ROM and provides a serial communication interface for debugging. Depending on the operating mode, the IO pads are multiplexed between the NPs and the NS430.

For testing, it is desirable to have direct access to an NP, bypassing the GCU and ADC FSM. To accomplish this, a lone NP exists outside of the NP grid, and its inputs and outputs driven and captured by DFFs in a scan chain. Input data is serially clocked into the scan chain and the resulting output data is captured by the chain as well, after which it is serially clocked out. A block diagram is shown in Figure 3.2.

3.5 Operating Modes

The chip features several operating modes which improve testability and multiplex the IO pads between several internal modules. In the primary mode of operation, the GCU and NPs operate independently with an external program ROM. Most of the

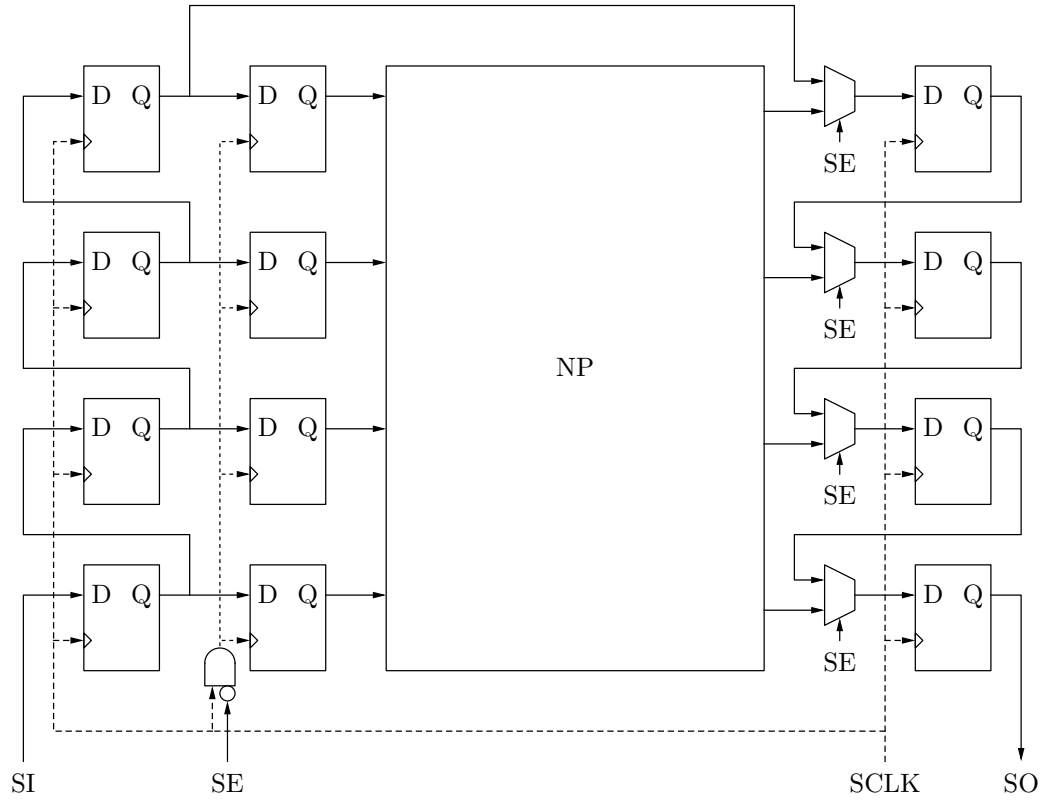


Figure 3.2: Block diagram of the scan chain NP test structure

IO pins are devoted to the address bus and data bus for the external ROM as well as the output data from the NPs. In a secondary mode, the internal NS430 takes control of the IO pins, which may be programmed to implement SPI, I2C and asynchronous serial interfaces as well as simple IO tasks. This mode is used when controlling the NPs internally with the NS430, but is also useful for testing the NS430 itself.

The ADC FSM may be bypassed and the control signals fed from outside the chip. If there is a design issue with the module, this allows the NPs to still capture images. It also provides a method for using longer reset and integration delays than allowed by the internal 16 bit counters. Also, nonlinear binary counters may be used to provide increased dynamic range.

Chapter 4

Synthesis

4.1 Overview

The RTL is synthesised to a gate-level netlist using Cadence RTL Compiler (RC). The netlist is translated to a 0.13 µm CMOS technology with 8 metal layers using a professional design kit that provides digital standard cells, pads and memories.

4.2 Timing Constraints

Timing constraints are applied at this stage in the design. They specify the target operating frequency, chip-level input and output (IO) delays as well as exceptions for nonstandard timing paths. RC uses timing constraints to select the drive strength of standard cells and to insert additional digital buffers to decrease signal propagation times as required to satisfy the constraints. Tighter constraints lead to larger buffers and a larger design area. The target area for the NPs leaves little spare area, so it is loosely constrained and the final timing significantly affects the maximum frequency of the system.

The distribution of global signals from the GCU to the NPs also limits the operating frequency. These signals route directly to the first row of NPs in the grid and then propagate down the columns, with each NP routing and buffering the signals to the next NP. If the propagation delay through an NP becomes large, it may limit the operating frequency. This delay is determined by the drive strength of the digital cells used to buffer the signals across the grid. There are many global signals, so each NP incorporates many of these buffers, so their drive strength is capped to maintain area requirements. Since the GCU is limited to controlling a 16×16 grid, the total propagation delay is bounded. Scaling beyond this would require multiple GCUs, so the propagation delay would not increase.

Practically delays associated with the internal logic of the NP dominate this propagation delay. In the final layout, approximately 250 ps is introduced for each additional row of NPs. This gives a total delay of 2 ns for the 8×10 grid of NPs compared to around 10 ns for delays internal to the NP. The slowest paths involve branch instructions where the GCU requires each NP to signal whether or not it should execute the jump. The reply is propagated back to the GCU across each column, where each NP generates the outbound reply by logically combining the inbound reply from the previous NP with its own reply. Due to this logic, more propagation time is required than for the global signals–about 650 ps per row of NPs. The total round-trip delay is the combination of propagation delay from the GCU through a column of NPs, memory access and comparison within the NP, and the delay from the reply propagating back through the column to the GCU.

External interfacing also limits the maximum system frequency. IO pads add several nanoseconds of delay. To minimize this delay, their drive strengths are increased while balancing the additional noise introduced to the power nets.

Finally the external ROM access time is also large, on the order of 10 ns for

fast asynchronous memories. This limitation is due to the simple, non-pipelined architecture and the lack of registered IO on the system boundaries.

Due to these delays, the final design operates at 20 MHz which accounts for external signal delays from the program ROM as well as internal propagation delays. For this design, the primary goal is to demonstrate the architecture, making the operating frequency is a secondary consideration. This may be increased by pipelining instruction fetching and data readout with no modifications to the NPs themselves. Also, since the design is implemented with standard cells, smaller technology nodes would use less area, allowing larger buffers to be used to reduce propagation delays through the NP.

4.3 Hierarchical Synthesis

By default, RC treats each NP in the grid as an independent design to allow it to be fully optimized based on its location in the chip. This leads to increased runtime when the size of the NP grid grows large and also causes routing problems later in the design flow. This implementation uses 80 NPs, so special consideration is necessary in order to complete the layout successfully and in reasonable time. Since each NP has similar timing requirements, that module is partitioned from the rest of the design and synthesized independently. The top-level netlist is then synthesized with the NPs modeled using abstract representation. Once complete, the NP partition is instantiated multiple times at the top-level to complete the hierarchy. This greatly reduces RC runtime at the cost of timing optimization, which is not critical since NPs are placed in a regular pattern.

4.4 Synthesis Results

The architectural changes described in Chapter 2 significantly reduce the required area for the NP compared to the original design from [15]. Table 4.1 compares the original area to an unoptimized version of the new RTL implementation as well as a final version that uses clock gating and smaller memory elements to further reduce area. Table 4.2 shows the original distribution of standard cell instances for each version, which reflects the removal of the tristate memory bus and an overall reduction in logic required to implement the new version.

Table 4.1: NP area in units of 1000 μm^2

	Original	Unoptimized	Optimized
Sequential	48.7	51.8	40.1
Inverter	2.7	0.1	0.2
Clock Gating	0.0	0.0	5.9
Tristate	6.2	0.0	0.0
Logic	26.9	16.1	15.4
Total	84.4	68.1	61.7

Table 4.2: NP instance counts

	Original	Unoptimized	Optimized
Sequential	1606	1639	1639
Inverter	618	42	59
Clock Gating	0	0	206
Tristate	537	0	0
Logic	3012	1973	1843
Total	5773	3654	3747

Chapter 5

Physical Layout

5.1 Overview

Cadence Encounter Digital Implementation (EDI) generates a layout by placing and routing the standard cells using the technology-specific netlist output from RC. Since this smart camera architecture is primarily digital, EDI generates the majority of the layout sent to the foundry for fabrication. Only the photodiodes and ADCs are hand-drawn in Cadence Virtuoso and inserted into the layout at stream-out. EDI uses a series of Tcl configuration scripts to control its behavior during layout generation.

EDI's default scripts work well for standard digital layouts that have large, unobstructed areas available for standard cell placement and routing. This design has very little area, so significant alterations are required. Routing is further complicated since the photodiodes and ADCs are embedded within the digital logic. Since metal must be kept clear of the light-sensitive photodiodes, regions in which signal wires may be routed and restricted and severe congestion occurs throughout the NP.

Similar to RC, EDI struggles to route designs with medium to large numbers of NPs in the imager. This is solved by partitioning the NP into its own subdesign and

routing it separately from the top-level. Once complete, the NP timing is characterized and stored in a model for use in top-level timing analysis. The rest of the chip is placed and routed using the NP model and then the top-level design and NP are streamed out as separate GDS2 files for fabrication. Each design undergoes a similar flow including floorplanning, standard cell placement, clock tree synthesis, signal routing, and design rule checks.

5.2 Metal Layer Allocation

The 0.13 µm process has 8 metal layers available for signal and power routing. The third-party standard cell library uses the lowest metal layer, metal 1, so it is largely unavailable for routing throughout the design. Metals 2 and 3 route digital signals between standard cells. These layers are the most congested in the NP design due to the dense, obstructed layout between photodiodes. The next highest layer, metal 4, exclusively contains digital power and ground stripes to minimize their resistance. Metal 5 acts as a low-impedance shield between the digital signals below and analog signals above that layer. Analog power and bias voltages for the photodiode ADCs use metal 6 stripes to route to each pixel. A voltage ramp required by the ADCs, generated externally, enters the chip through a pin and branches out in an H tree pattern across the array on metal 7. Additional analog power and ground stripes use metal 8 to reduce the resistance of those nets.

5.3 Power Grid

There are four supplies in the chip that are used for digital IO, digital power for core standard cells, analog power and a low-noise power supply used for shielding.

Additional bias voltages for the analog sections are also present. With the exception of the IO supply which does not enter the core region of the chip, all of these supplies use a series of rings and stripes to reach the analog blocks in each pixel. Area constraints require these to be tightly layered summarized in Table 5.1. Figure 5.1 shows the layout of the power stripes and shielding between photodiodes.

Table 5.1: Signals distributed to photodiodes through the power grid

Name	Routing Layer
Digital Supply	Metals 1 and 4
Shield Supply	Metal 5
ADC Bias Voltages	Metal 6
ADC Voltage Ramp	Metal 7
Analog Supply	Metals 6 and 7

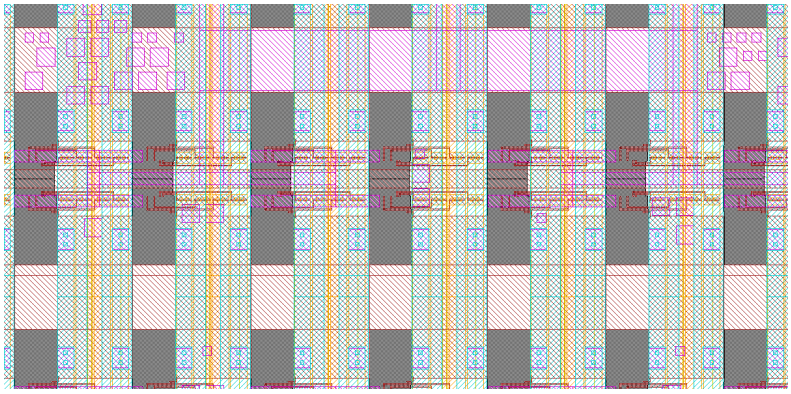


Figure 5.1: Layout of power stripes and analog bias voltages between pixels

Vias used to connect digital power stripes on metal 4 to standard cell power rails on metal 1 are large enough to block signal routing on the intermediate layers. To help congestion, the layout uses fewer vias while preserving redundancy in order to improve yield. The constraints on stripe width and length along with reduced via counts increases the resistance of the power grid. The voltage drop of the grid was analyzed using EDI and was insignificant with at most a 3 mV drop. Figure 5.2 shows a visualization of the voltage drop across the chip.

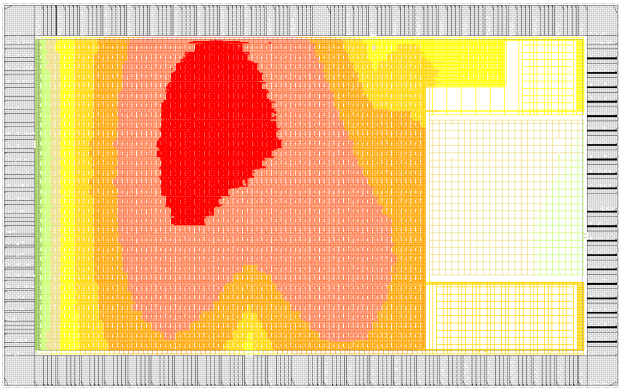


Figure 5.2: Voltage drop in the power grid from 0 mV (green) to 3 mV (red)

5.4 Input and Output Pads

Using a Tcl script, EDI places a ring of IO pads, taken from the netlist output from RC, around the boundary of the chip. These provide wire bonding sites to connect circuits on the silicon die to package pins. After fabrication of the chip, bond wires are added during packaging between these pads and the pins. The angle the bond wire makes with the die must be greater than 45° according to the manufacturer. Since the die is not square, 4 of the 100 package pins are left disconnected to satisfy this requirement. Figure 5.3 demonstrates the issue for a package with 20 pins. The pad outlined with a dotted line shows the best attainable position for a hypothetical pad connected to pin 1. Since it is located outside of the pad ring, it is not possible to include it in the chip.

5.5 Analog Routing

Each pixel contains a light-sensitive photodiode along with an ADC and is designed separately from EDI in Cadence Virtuoso. EDI uses a simplified, abstract version of this layout which contains pin locations for routing. The photodiodes are equally spaced 39.6 µm apart to form a 64×80 resolution imager. The analog area is shared

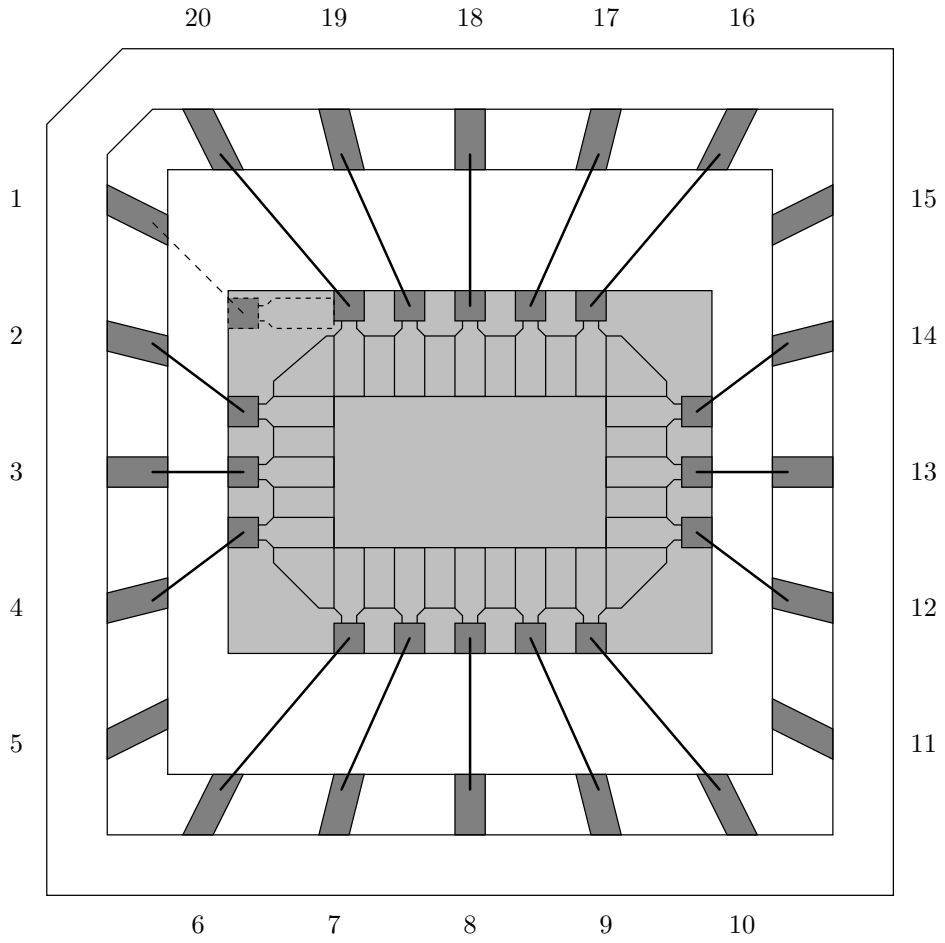


Figure 5.3: Corner pins left unconnected due to bond wire angle requirements

between two pixels by mirroring every other row, shown in Figure 5.4. This doesn't save significant area, but grouping analog blocks leaves large regions open for digital routing, reducing congestion.

Analog signals enter the design from outside the chip and are globally routed to the photodiode ADCs. DC bias voltages are distributed with rings and stripes similar to the power and ground rails. A Tcl script programmatically routes the pins on each photodiode to the corresponding stripe. The photodiodes also require an analog ramp with minimal skew between each pixel. This net is routed with an H tree-like fractal shown in Figure 5.5 to minimize skew. Since the NP grid is rectangular instead of

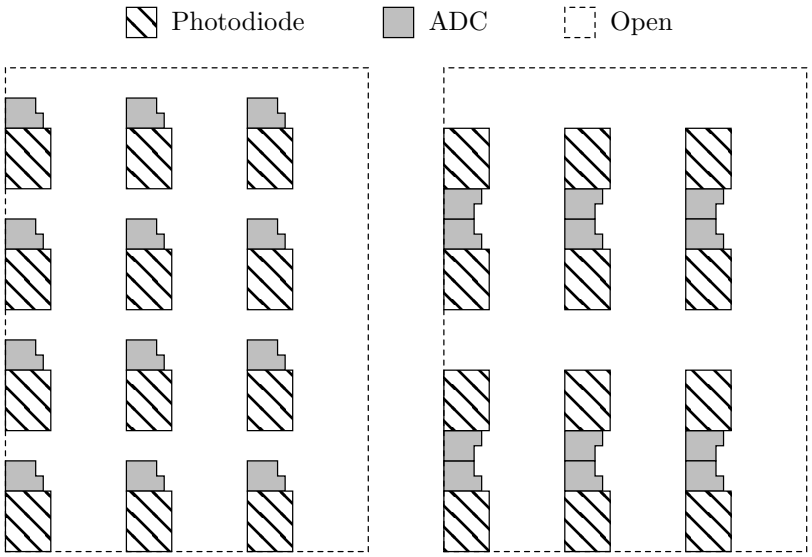


Figure 5.4: Mirrored photodiodes for larger unobstructed regions

square, the pattern is modified with additional accordion routes to maintain equal delays to the endpoints.

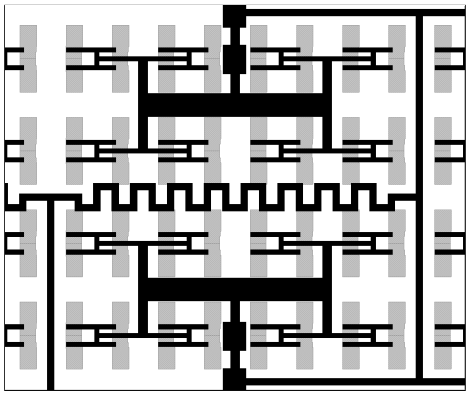


Figure 5.5: H tree network used to distribute ADC ramp voltage with low skew.

The analog nets use the upper metal layers for routing to physically separate them from noisy digital nets on lower metal layers. An intermediate metal layer is used to form a grounded shield between the digital and analog metal layers for improved noise performance. This shield is isolated from the digital and analog ground nets and is design to be connected to the ground network externally to reduce ground bounce.

5.6 Standard Cell Placement

Careful standard cell placement is critical to prevent routing congestion later in the flow. In general, cells in the same design module should be located near one another to minimize the length of the metal interconnects. EDI does this by default, but this can lead to routing issues when the standard cells have relatively high numbers of pins relative to their area. This occurs primarily with designs containing complex combinational logic, specifically the NP ALU. During placement, empty space is added between the standard cells in the ALU, which loosens the routing requirements in this region. With no standard cells in these regions, metal 1 is available for routing, which reduces congestion.

5.7 Clock Tree Synthesis

EDI accepts timing specifications to perform clock tree synthesis (CTS) in digital designs. Low skew, low delay clock distribution across the NP grid is challenging due to the chip's large dimensions. As before, a hierarchical approach provides a consistent and efficient solution to this problem. First a clock tree is synthesized for a master NP design and then cloned repeatedly to form the NP grid. The timing specification for the NP clock tree is relatively lax to prevent EDI from using large area for additional clock buffers. Figure 5.6 shows a histogram of the clock delays to each endpoint and Figure 5.7 shows the relative phase delays.

The skews and delays from this CTS are propagated to the top-level where they are used during top-level CTS. At the top-level, routing channels between adjacent NPs, shown in Figure 5.8, provide enough open area to route the clock tree to each NP unobstructed by other signal routes. This establishes a clock tree with significant,

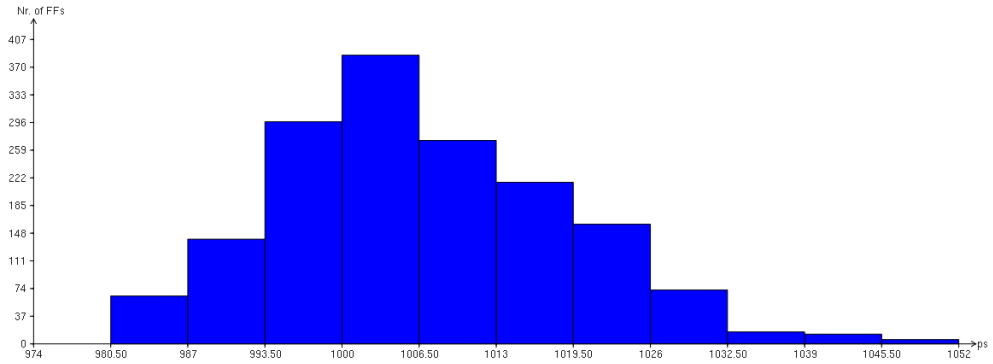


Figure 5.6: Histogram output from EDI's CTS debugging tool showing the phase delay of the clock inside the NP

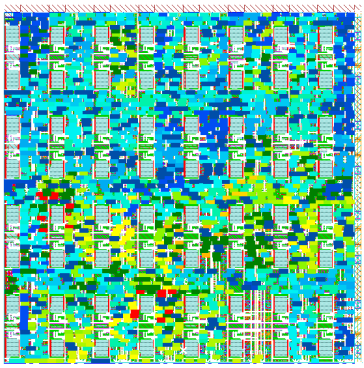


Figure 5.7: Early (blue) and late (red) NP clock phase delay at each termination

but acceptable skew and insertion delay. The final clock skew and delay is shown in the histogram in Figure 5.9. Figure 5.10 shows the phase delays between the NPs in the grid and the GCU.

5.8 Digital Routing

EDI performs timing-driven signal routing which automatically resizes standard cells and inserts additional buffering to ensure timing is met. It uses the same timing constraints from RC for this purpose. To account for process, voltage and temperature variations, EDI loads multiple timing libraries which characterize the standard cells

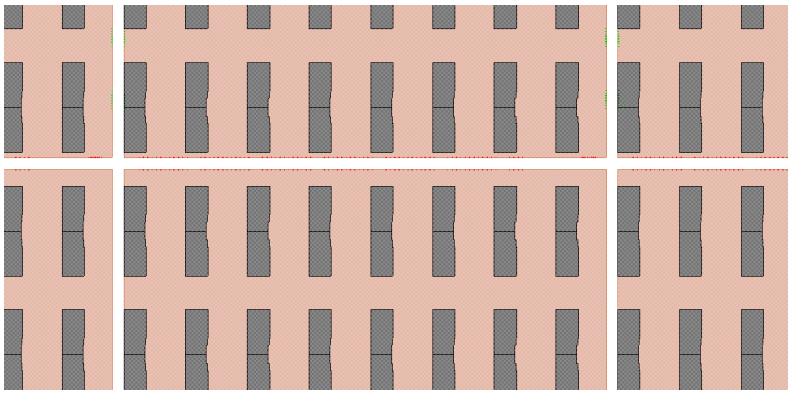


Figure 5.8: Channels between NPs that provide room to route the top-level clock

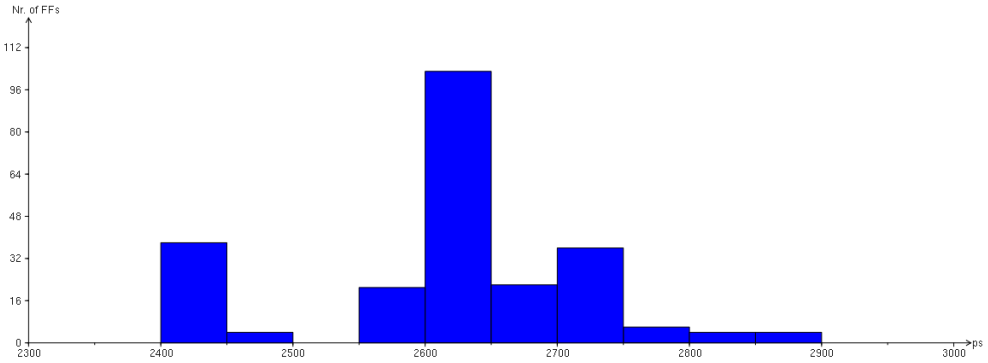


Figure 5.9: Histogram output from EDI's CTS debugging tool showing the phase delay of the clock for the top-level design

in each operating region. Setup and hold timing violations are checked using the most pessimistic timing library available, which improves timing compliance after fabrication. The distribution of signal setup slack across the chip is shown in Figure 5.11.

5.9 Signoff and Export

With routing complete, EDI performs design rule checks (DRC) to verify that the chip meets fabrication requirements. These are determined by the foundry where the chip is being manufactured. Finally the tool exports the layout as a GDS2 file, an

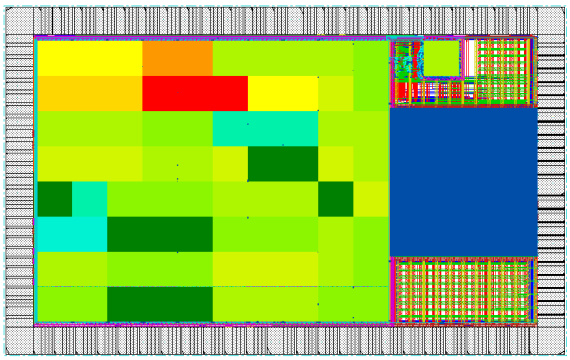


Figure 5.10: Early (blue) and late (red) top-level clock phase delay at each termination

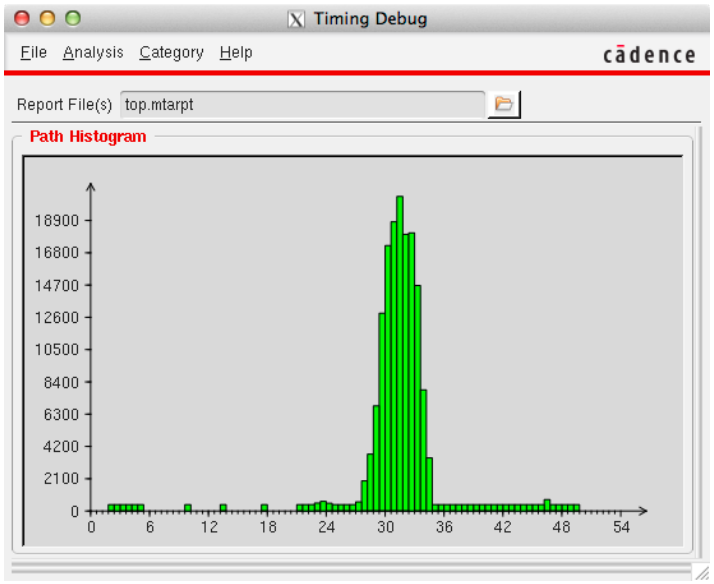


Figure 5.11: Distribution of signal slack in design

industry-standard format for chip layouts. Cadence Virtuoso then imports this file for integration with the analog design blocks.

The layout of each NP, shown in Figure 5.12, is 316.8 µm × 316.8 µm with a pixel pitch of 39.6 µm. Figure 5.13 shows the chip-level layout which contains an estimated 2 million transistors. It measures 5.070 mm × 3.155 mm with a total area of 16 mm^2.

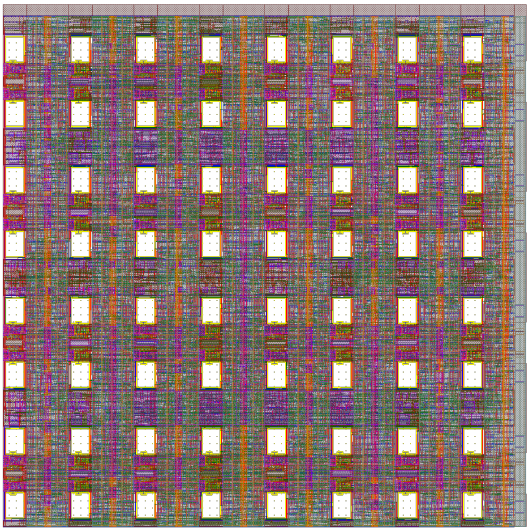


Figure 5.12: NP layout

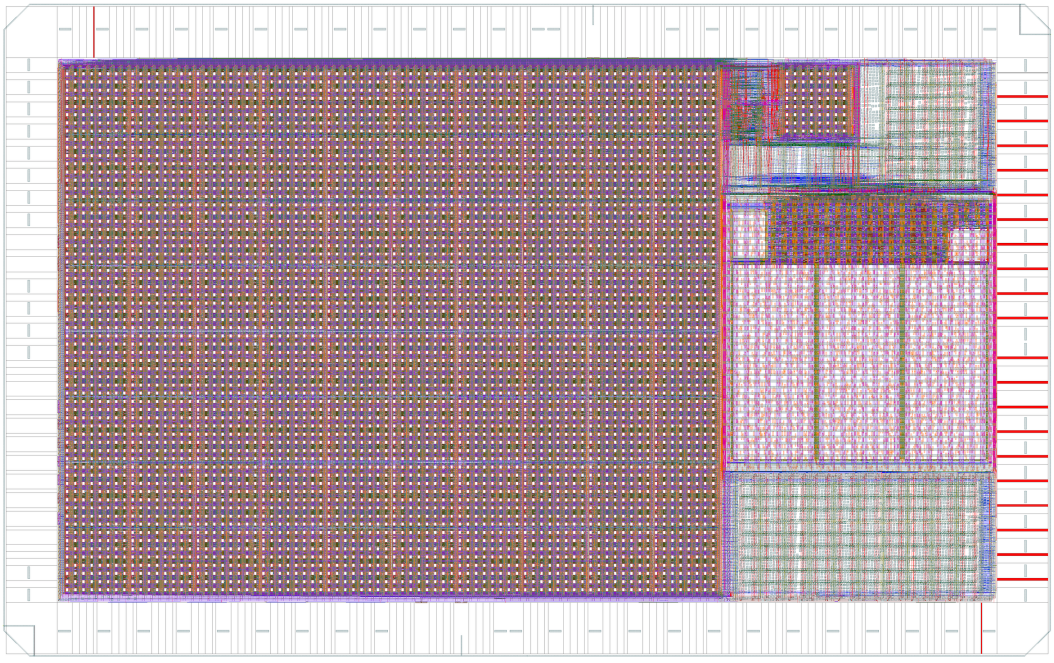


Figure 5.13: Chip layout

Chapter 6

Simulation and Testing

6.1 Assembly Language Programming

A major advantage of the NP architecture is its non-specialized instruction set that accommodates a wide range of algorithms. Writing such algorithms requires a simple, convenient programming language. For simplicity, NPs use a language with syntax inspired by Intel and ARM assembly. It provides access to all available instructions and addressing modes implemented in the architecture. A custom command-line assembler translates code into ROM binaries as well as other formats useful for testing the design on computer and FPGA. A full list of output options is shown in Appendix B.1. The assembler implements additional programming conveniences including assembly-time expression evaluation for constants, address resolution of program labels and macro expansion with variables.

6.2 FPGA Testing

An Xilinx Virtex 5 FPGA development board served as the platform for the initial testing of the revised NP architecture. It housed all necessary system modules including a small 3×4 grid of NPs, the GCU and ROM code stored by FPGA block RAMs. A simple controller provided a serial interface that allowed read and write access to the program ROM. It could update the ROM at any time, removing the need to reconfigure the FPGA during software debugging. Without access to physical ADCs, NPs used preloaded image data during testing. After each algorithm, the serial controller recorded and transmitted the result to a host computer for verification. The FPGA test setup performed several simple image operations to verify the fundamental functionality of the architecture. These include image translations and edge detection similar to those performed in [15].

6.3 Post-Layout Simulation

After EDI generates a final layout netlist along with annotations of the timing delays, Cadence Incisive Simulator (NCSIM) uses these files to perform an accurate functional simulation of the final chip. The timing delays are applied to the netlist during simulation, which models the propagation delays through gates and over long interconnects, including clock delay and skew. In order to exercise the NP architecture, the full 8×10 grid of NPs execute a variant of the EZW compression algorithm from [10] and [11] adapted to the architecture.

In initial testing, the NPs loaded image data stored as program constants from the external ROM, similar to what would happen on a test PCB populated with the packaged chip. The resulting output is captured from the column data bus, written

to a text file and then parsed by scripts into a verifiable format. A second simulation tested the ADC image capture logic by simulating the analog interface. To accomplish this, an input image as converted to corresponding time delays normally generated by the analog electronics. During the simulation, the analog outputs were forcibly toggled based on these calculated delays, similar to what occurs during an actual image capture. The resulting output was verified against reference outputs. The scan chain NP test structure was also exercised. To verify basic functionality, a short program was shifted in one instruction at a time, executed and shifted out. The resulting output verified it can be controlled externally.

Chapter 7

Conclusion and Future Work

7.1 Future Work

7.1.1 NP Area Reduction

In order to obtain better image quality and resolution, a reduction in area consumed by the digital standard cells is required. This will lead to higher resolution imagers with a better fill factor. The NPs final dimension is limited by the available area for digital standard cells and the number of open tracks to route metal interconnects. Outside of the analog photodiodes which consume approximately 25% of the pixel area, NPs require no custom circuitry and consist of digital standard cells. This means the architecture scales significantly and easily with shrinking CMOS technology nodes which halve in size approximately every two years. To mitigate routing congestion, these technologies also offer large numbers of routing layers which were not available in this design.

7.1.2 Improved Pixel Memory

As seen in Table 4.1, the NP memory consumes about 75% of the digital area. This is implemented using D flip-flops (DFF) since they allow for flexible placement around the photodiodes in the design. However, they consume a large amount of area compared to other standard cells. Alternative memory types offer higher densities while still being small enough to fit between pixels. Some digital standard cell libraries offer latch-based register file cells, although these require additional care during timing analysis. Even smaller, 24-bit DRAMs could be used if they were custom designed, as in [2, 14].

7.1.3 Power Consumption

The primary focus of this work is to implement and test the NP architecture, leaving significant improvements in the power consumption performance of the design for a future iteration. While individual NPs may be turned off to reduce switching current, the clock tree still toggles and consumes significant switching power due to its high fanout. Additional clock gating at the entry point of the clock into each NP would remove this switching current. The neighborhoods are ideal for power supply switching as well. Use of high-side switching cells would allow the power grid in entire regions of a chip to be powered down, reducing leakage currents.

7.1.4 Switchable ADC Bias

Currently the analog blocks in each pixel are powered continuously even when not in use. The ADC FSM, which already provides counters and control signals for the pixel ADCs, could be expanded to dynamically power the ADCs shortly before capturing an image and to turn them off during idle periods.

7.2 Conclusion

A new, neighborhood-based smart camera architecture is implemented using a 0.13 µm, 8 metal CMOS process using design tools from Cadence. It possesses the scalability of pixel-level processing elements as well as the programming flexibility of chip-level processors. The architecture is first realized in VHDL and tested on an FPGA, after which significant scripting and configuration allows RC and EDI to generate a DRC-clean layout suitable for fabrication. Post-layout simulation with back-annotated timing with NCSIM verifies the functionality of the final layout.

Appendix A

Instruction Set Reference

Name	Binary	Description
ADC	0000 1DDS SSSS SSSS	Add with carry
SUB	0001 0DDS SSSS SSSS	Subtract
SBC	0001 1DDS SSSS SSSS	Subtract with carry
AND	0010 0DDS SSSS SSSS	Bitwise AND
ORR	0010 1DDS SSSS SSSS	Bitwise OR
EOR	0011 0DDS SSSS SSSS	Bitwise exclusive OR
CMP	0011 1DDS SSSS SSSS	Compare
LDR	0100 0DDS SSSS SSSS	Load register
STR	0100 1DDS SSSS SSSS	Store register
ASR	1000 1DD1 000- ----	Arithmetic shift right into carry
LSL	1000 1DD0 100- ----	Logical shift left into carry
LSR	1000 1DD0 010- ----	Logical shift right into carry
GTB	1000 1DD0 001- ----	Convert gray code to binary code
BEQ	110A AAAA AAAA 0000	Branch if equal
BNE	110A AAAA AAAA 0001	Branch if not equal
BCS/BHS	110A AAAA AAAA 0010	Branch if carry set/higher or the same (uns)
BCC/BLO	110A AAAA AAAA 0011	Branch if carry clear/lower (uns)
BMI	110A AAAA AAAA 0100	Branch if negative
BPL	110A AAAA AAAA 0101	Branch if positive or zero
BVS	110A AAAA AAAA 0110	Branch if overflow set
BVC	110A AAAA AAAA 0111	Branch if overflow clear
BHI	110A AAAA AAAA 1000	Branch if higher (uns)
BLS	110A AAAA AAAA 1001	Branch if lower or same (uns)
BGE	110A AAAA AAAA 1010	Branch if greater than or equal (sgn)
BLT	110A AAAA AAAA 1011	Branch if less than (sgn)
BGT	110A AAAA AAAA 1100	Branch if greater than (sgn)
BLE	110A AAAA AAAA 1101	Branch if less than or equal (sgn)
BAL/B	110A AAAA AAAA 1111	Branch always
BL	111A AAAA AAAA ----	Branch and link PC+1 to GCU stack
ZEQ	1001 ---- ---- 0000	Sleep if equal
ZNE	1001 ---- ---- 0001	Sleep if not equal
ZCS/ZHS	1001 ---- ---- 0010	Sleep if carry set/higher or the same (uns)
ZCC/ZLO	1001 ---- ---- 0011	Sleep if carry clear/lower (uns)
ZMI	1001 ---- ---- 0100	Sleep if negative
ZPL	1001 ---- ---- 0101	Sleep if positive or zero
ZVS	1001 ---- ---- 0110	Sleep if overflow set
ZVC	1001 ---- ---- 0111	Sleep if overflow clear
ZHI	1001 ---- ---- 1000	Sleep if higher (uns)
ZLS	1001 ---- ---- 1001	Sleep if lower or same (uns)
ZGE	1001 ---- ---- 1010	Sleep if greater than or equal (sgn)
ZLT	1001 ---- ---- 1011	Sleep if less than (sgn)
ZGT	1001 ---- ---- 1100	Sleep if greater than (sgn)
ZLE	1001 ---- ---- 1101	Sleep if less than or equal (sgn)
ZAL/Z	1001 ---- ---- 1111	Sleep always
RST	1000 0011 0000 ----	Clear sleep history and flags
NOP	1000 0010 1000 ----	No operation
WAK	1000 0010 0100 ----	Wake all NPs up by one level.
IMG	1000 0010 0010 ----	Acquire a new sample for the ADC
BX	1000 0010 0001 ----	Branch and exchange
ROM	1000 0001 BB-- ----	Switch program ROMs (limited to 512 instructions each)
OUT	101W WWWS SSSS SSSS	Output data source on column data bus
TINTL	1000 0100 LLLL LLLL	Store the low byte of the ADC integration time register.
TINTH	1000 0101 HHHH HHHH	Store the high byte of the ADC integration time register.
TRSTL	1000 0110 LLLL LLLL	Store the high byte of the ADC reset time register.
TRSTH	1000 0111 HHHH HHHH	Store the low byte of the ADC reset time register.

R:*row* D:*destination register* L:*low byte* I:*immediate value* W:*output row*
C:*column* S:*data source* H:*high byte* A:*program address* B:*ROM bank*

Table A.1: Instruction Set

Name	Binary	Description
X	`0 00RR RCCC`	X Memory
Y	`0 01RR RCCC`	Y Memory
Z	`0 10RR RCCC`	Z Memory
V0	`0 1100 0000`	Variable 0
V1	`0 1100 0001`	Variable 1
V2	`0 1100 0010`	Variable 2
V3	`0 1100 0011`	Variable 3
V4	`0 1100 0100`	Variable 4
V5	`0 1100 0101`	Variable 5
V6	`0 1100 0110`	Variable 6
V7	`0 1100 0111`	Variable 7
R0	`0 1101 0000`	Register 0
R1	`0 1101 0001`	Register 1
R2	`0 1101 0010`	Register 2
R3	`0 1101 0011`	Register 3
N	`0 1110 0001`	Data from the NP to the north
S	`0 1110 0010`	Data from the NP to the south
W	`0 1110 0100`	Data from the NP to the west
E	`0 1110 1000`	Data from the NP to the east
R0 Indirect	`0 1111 0000`	Data from the XYZ memory addressed with R0
R1 Indirect	`0 1111 0001`	Data form the XYZ memory addressed with R1
R2 Indirect	`0 1111 0010`	Data form the XYZ memory addressed with R2
R3 Inidrect	`0 1111 0011`	Data form the XYZ memory addressed with R3
SR	`0 1111 0100`	Status register
RCR	`0 1111 1000`	Row/column register
IMM	`1 IIII IIII`	Immediate value

`R`:*row* `D`:*destination register* `L`:*low byte* `I`:*immediate value* `W`:*output row*
`C`:*column* `S`:*data source* `H`:*high byte* `A`:*program address* `B`:*ROM bank*

Table A.2: Data Sources

Appendix B

Toolchain Commands

B.1 NP Assembler

```
usage: npasm [-h] [-a] [-b] [-l] [-n name] [-o outdir] [-p] [-r] [-v] infile

Generates binary, VHDL, and other useful output from an input neighborhood
processor assembly file.

positional arguments:
  infile                Path to the input assembly file.

optional arguments:
  -h, --help            show this help message and exit
  -a, --ascii           Save an ascii file of the binary program instructions.
  -b, --bin             Save a binary file containing the program
                        instructions.
  -l, --list            Save a listing file containing the original assembly,
                        instruction, and memory location in one document.
  -n name, --name name  Specify a filename for the output files. Defaults to
                        the input filename.
  -o outdir, --out outdir
                        Specify path to store the output files in. Defaults to
                        the working directory.
  -p, --pre             Save the output of the preprocessor after #define
                        statements are populated and numeric arguments are
                        converted.
  -r, --rtl             Save a VHDL file with a ROM containing the program
                        instructions.
  -v, --verbose         The amount of output to display in the terminal.
                        Defaults to no output, and increases with additional
                        -v, e.g. -vvvvv.
```

Appendix C

Packaging and Pinout

Table C.1: Package pinout and IO buffers.

Pin #	Side	Pad Name
1	West	not connected
2	West	pad_col_sel_0
3	West	pad_col_sel_1
4	West	pad_col_sel_2
5	West	pad_col_sel_3
6	West	pad_adc_ramp_rst
7	West	pad_adc_rst
8	West	pad_reg_en
9	West	pad_active
10	West	pad_vdd_w
11	West	pad_gnd_w
12	West	pad_mode_0
13	West	pad_mode_1
14	West	pad_dgnd_w
15	West	pad_dvdd_w
16	West	pad_agnd
17	West	pad_qgnd
18	West	pad_adc_vrst
19	West	pad_adc_vramp
20	West	pad_adc_vbias
21	West	pad_qvdd

continued on the following page

Pin #	Side	Pad Name
22	West	pad_avdd
23	West	pad_gpin_0
24	West	pad_gpout_0
25	West	not connected
26	South	pad_adc_bypass
27	South	pad_adc_counter_0
28	South	pad_adc_counter_1
29	South	pad_adc_counter_2
30	South	pad_adc_counter_3
31	South	pad_adc_counter_4
32	South	pad_adc_counter_5
33	South	pad_adc_counter_6
34	South	pad_adc_counter_7
35	South	pad_vdd_s
36	South	pad_gnd_s
37	South	pad_pir_clk_in
38	South	pad_pir_clk_out
39	South	pad_dgnd_s
40	South	pad_dvdd_s
41	South	pad_scnp_sclk
42	South	pad_scnp_se
43	South	pad_scnp_si
44	South	pad_scnp_so
45	South	pad_dff_clk
46	South	pad_dff_div4
47	South	pad_dff_div8
48	South	pad_dff_a
49	South	pad_dff_b
50	South	pad_dff_c
51	East	not connected
52	East	pad_ir_00
53	East	pad_ir_01
54	East	pad_ir_02
55	East	pad_ir_03
56	East	pad_ir_04
57	East	pad_ir_05
58	East	pad_ir_06
59	East	pad_ir_07
60	East	pad_vdd_e
61	East	pad_gnd_e

continued on the following page

Pin #	Side	Pad Name
62	East	pad_msp_lfclk
63	East	pad_msp_lfclk
64	East	pad_dgnd_e
65	East	pad_dvdd_e
62	East	pad_msp_lfclk
67	East	pad_ir_09
68	East	pad_ir_10
69	East	pad_ir_11
70	East	pad_ir_12
71	East	pad_ir_13
72	East	pad_ir_14
73	East	pad_ir_15
74	East	pad_msp_rst
75	East	not connected
76	North	pad_pc_0
77	North	pad_pc_1
78	North	pad_pc_2
79	North	pad_pc_3
80	North	pad_pc_4
81	North	pad_pc_5
82	North	pad_pc_6
83	North	pad_pc_7
84	North	pad_pc_8
85	North	pad_vdd_n
86	North	pad_gnd_n
87	North	pad_msp_hfclk
88	North	pad_msp_hfclk
89	North	pad_dgnd_n
90	North	pad_dvdd_n
91	North	pad_col_data_0
92	North	pad_col_data_1
93	North	pad_col_data_2
94	North	pad_col_data_3
95	North	pad_col_data_4
96	North	pad_col_data_5
97	North	pad_col_data_6
98	North	pad_col_data_7
99	North	pad_col_we
100	North	pad_pir_rst

Bibliography

[1] E.R. Fossum. CMOS image sensors: electronic camera-on-a-chip. *Electron Devices, IEEE Transactions on*, 44(10):1689–1698, 1997.

[2] S. Kleinfelder, S. Lim, X. Liu, and A. El Gamal. A 10,000 frames/s 0.18 μm CMOS digital pixel sensor with pixel-level memory. In *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, pages 88–89, 2001.

[3] T. Komuro, S. Kagami, and M. Ishikawa. A dynamically reconfigurable SIMD processor for a vision chip. *Solid-State Circuits, IEEE Journal of*, 39(1):265–268, 2004.

[4] Q. Lin, W. Miao, W. Zhang, Q. Fu, and N. Wu. A 1,000 Frames/s Programmable Vision Chip with Variable Resolution and Row-Pixel-Mixed Parallel Image Processors. *Sensors (Basel)*, 9(8):5933–5951, 2009.

[5] H. Zhu and T. Shibata. A real-time image recognition system using a global directional-edge-feature extraction VLSI processor. In *ESSCIRC, 2009. ESSCIRC '09. Proceedings of*, pages 248–251, 2009.

[6] W. Zhang, Q. Fu, and N. Wu. A programmable vision chip based on multiple levels of parallel processors. *Solid-State Circuits, IEEE Journal of*, 46(9):2132–2147, 2011.

[7] A. Verdant, A. Dupret, P. Villard, L. Alacoque, H. Mathias, and F. Delgehier. A 120 μw 240 $\times$ 110 @ 25fps vision chip with ROI detection SIMD processing unit. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 2412–2415, 2013.

[8] H. Zhu and T. Shibata. A real-time motion-feature-extraction image processor employing digital-pixel-sensor-based parallel architecture. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 1612–1615, 2012.

[9] H. Yamasaki and T. Shibata. A real-time image-feature-extraction and vector-generation VLSI employing arrayed-shift-register architecture. *Solid-State Circuits, IEEE Journal of*, 42(9):2046–2053, 2007.

[10] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *Signal Processing, IEEE Transactions on*, 41(12):3445–3462, 1993.

[11] A. Said and W.A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):243–250, 1996.

[12] M. Yagi, M. Adachi, and T. Shibata. A hardware-friendly soft-computing algorithm for image recognition.

[13] K. Sayood. *Introduction to Data Compression*. Elselvier, 225 Wyman Street, Waltham, MA, 4th edition, 2012.

[14] K. Ito, B. Tongprasit, and T. Shibata. A computational digital pixel sensor featuring block-readout architecture for on-chip image processing. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(1):114–123, 2009.

[15] A. Nelliparthi. A 2-D processor array for massively parallel image processing. Master's thesis, University of Nebraska-Lincoln, Lincoln, Nebraska, December 2011.